Шаблоны функций и классов

Языки программирования С и Си++ обеспечивают строгую проверку типов данных. Некоторые языки не обеспечивают такой проверки и она полностью ложится на плечи программиста. Например в языке PL1 вы можете сравнивать значение строковой и числовой переменных. Это не будет ошибкой с точки зрения компилятора. Если вы случайно допустите ошибку, то обнаружить ее будет достаточно сложно.

Однако строгая типизация, присущая некоторым языкам, не всегда удобна. Если вам надо выполнять одинаковые вычисления над переменными различных типов, придется создавать две фактически одинаковые функции, которые оперируют с различными типами данных.

Аналогичная ситуация возникает, если вам надо создать класс, для работы с различными типами данных. Допустим вам надо создать список из элементов. Удобнее всего это сделать при помощи классов. Но если вам надо, чтобы в качестве элементов списка фигурировали различные типы данных, то вам наверняка придется написать несколько одинаковых классов, различающихся только типом элементов.

Естественно такая ситуация усложняет работу программиста, увеличивают объем исходных текстов программы, и наконец просто может стать причиной ошибок и несоответствий. Так, если в вашей программе содержится несколько функций или классов, отличающихся только типом данных, которыми они оперируют, то когда вы вносите исправления в одну функцию, надо будет точно также исправлять и все остальные.

Чтобы облегчить программисту работу, в стандарт языка Си++ было включено понятие шаблона. В Visual C++ шаблоны реализованы начиная с версии 2.0. Ранние версии Visual C++ с ними работать не могли.

Шаблоны предназначены для создания ряда классов и функций, отличающихся только типом обрабатываемых ими данных. Для определения шаблона предназначено ключевое слово template (шаблон). Общий синтаксис определения шаблона имеет следующий вид:

template <template-argument-list> declaration;

Аргумент template-argument-list представляет собой список условных имен для определения типов, по которым будут различаться различные реализации классов или функций данного шаблона.

Понятие шаблонной функции. Общая форма объявления шаблонной функции. Ключевые слова template, class, typename

Понятие шаблона в функции имеет в виду использование обобщенного типа данных в качестве входных и выходных параметров функции. Вместо конкретного типа данных в своей работе функция использует некоторый обобщенный тип, который носит обобщенное имя, например T.

Общая форма обобщенной функции имеет вид:

*template <class T> return_type **FunName**(list_of_parameters)*
{
* // ...*
}
где
return_type – тип, который возвращает функция;
Fun_Name – имя шаблонной функции;
list_of_parameters – параметры шаблонной функции, которые имеют обобщенный тип T.

При объявлении шаблонной функции, вместо ключевого слова class можно применять ключевое слово typename. В этом случае общая форма обобщенной функции будет иметь вид:

```
template <typename T> return_type FunName(list_of_parameters)
{
   // ...
}
```

Например. Объявление функции, которая получает входным параметром переменную обобщенного типа T и возвращает значение обобщенного типа T имеет вид:

```
template <class T>

T FunName(T parameterName)
{
   // тело функции
   // ...
}
```

где

FunName – имя обобщенной функции;

T – имя обобщенного типа. При объявлении шаблонной функции, это имя может быть любым, например TYPE;

parameterName – имя параметра обобщенного типа T, который используется в функции FunName.

Пример 1. В примере реализована шаблонная функция Add(), которая добавляет два параметра обобщенного типа TT. В функции main() продемонстрировано использование функции Add()

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// Шаблонная функция, возвращающая сумму двух величин типа TT
template <class TT>
TT Add(TT t1, TT t2)
{
   return t1 + t2;
}

int main()
{
   // Использование шаблонной функции Add
   // 1. Для типа double
   double d1 = 3.85, d2 = 2.50;
   double d3;
   d3 = Add(d1, d2); // d3 = 6.35

   // 2. Для типа int
   int i1 = 25;
   int i2 = 13;
   int i3;
   i3 = Add(i1, i2); // i3 = 38

   // 3. Для типа string
   string s1 = "abc";
```

```
   string s2 = "def";
   string s3 = Add(s1, s2); // s3 = "abcdef"

   return 0;
}
```
Как видно из примера, шаблонная функция
```
template <class TT>
TT Add(TT t1, TT t2)
{
   return t1 + t2;
}
```
используется для суммирования параметров типов double, int, string. В этой функции обобщенный тип носит имя TT.

Пример 2. Реализована шаблонная функция Equal(), которая сравнивает два входных параметра обобщенного типа X. Функция возвращает true, если значения параметров одинаковы. В противном случае функция возвращает false.
```
#include "stdafx.h"
#include <iostream>
using namespace std;

// функция, которая сравнивает два входных параметра типа X
template <typename X>
bool Equal(X a, X b)
{
   return a==b;
}

int main()
{
   // демонстрация шаблонной функции Equal()
   bool res;

   res = Equal(5, 6); // res = 0
   res = Equal("abcd", "abcd"); // res = 1
   res = Equal(5.5, 5.5); // res = 1
   res = Equal('A', 'B'); // res = 0

   return 0;
}
```
Как видно из примера, вызов функции Equal() осуществляется для разных типов: int, string, double, char.

Если компилятор встречает шаблонную функцию, он генерирует версии функции для любого варианта ее применения с конкретным типом. То есть, если шаблонная функция вызывается для типов double, int, string, то компилятор генерирует 3 версии функции для каждого типа.

Например. Пусть задана функция, которая суммирует два параметра обобщенного типа T:
```
T Add(T t1, T t2)
{
   return t1 + t2;
}
```
При следующем вызове такой функции

```
int a = Add(3, 8);
double d = Add(3.8, 2.9);
string s = Add("Hello ", "world!");
```

компилятор сгенерирует следующие реализации функции Add():

```
int Add(int t1, int t2)
{
   return t1 + t2;
}

double Add(double t1, double t2)
{
   return t1 + t2;
}

string Add(string t1, string t2)
{
   return t1 + t2;
}
```

Вывод: компилятор генерирует столько вариантов функции, сколько существует способов ее вызова в программе.

Если нужно, в шаблонной функции может быть определено любое количество обобщенных типов (два, три и т.д.). В этом случае обобщенные типы перечисляются через запятую с помощью ключевого слова class или typename.

Например. Для двух типов с именами T1, T2 шаблонная функция будет иметь следующий общий вид

```
template <class T1, class T2>
return_type FunName(list_of_parameters)
{
  // ...
}
```

где

return_type – тип, который возвращает функция;
Fun_Name – имя шаблонной функции;
list_of_parameters – параметры шаблонной функции, которая оперирует обобщенными типами T1, T2.

Пример шаблонной функции, которая использует два обобщенных типа T1, T2. Функция получает два параметра типов T1, T2 и выводит значение этих параметров на экран. Для объявления типов используется ключевое слово typename. Можно также использовать ключевое слово class.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// Шаблонная функция, которая выводит значения величин t1, t2
template <typename T1, typename T2>
void Display(T1 t1, T2 t2)
{
   cout << "t1 = " << t1 << "; t2 = " << t2 << endl;
}

int main()
{
```

```
    int a = 30;
    double d = 9.83;
    char c = 'X';
    bool b = true;

    Display(a, d); // t1 = 30; t2 = 9.83
    Display(b, c); // t1 = 1; t2 = X
    Display(8, -100); // t1 = 8; t2 = -100

    return 0;
}
```

Явная «перегрузка» шаблонной функции.

Явная «перегрузка» (explicit specialization) шаблонной функции – это объявление еще одной функции с таким же именем, но уже для конкретного типа (например, int, double). Таким образом, «перегруженная» функция замещает шаблонную (обобщенную) функцию для некоторого конкретного случая (типа данных). Для других случаев вызывается обобщенный вариант шаблонной функции.

Пример. В примере шаблонная функция Display() «перегружена» для типа string. Такая «перегрузка» есть логически правильной, поскольку в операторе cout вывести тип string обычным способом

```
cout << "t = " << t << endl;
```

не удастся. В этом случае выйдет ошибка компиляции, поскольку для типа string нужно преобразование t.c_str() в тип const char *. Такое преобразование реализовано в «перегруженной» функции Display(), которая получает параметром переменную t типа string.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// шаблонная функция для типа T
template <class T>
void Display(T t)
{
    cout << "t = " << t << endl;
}

// явно "перегруженная" функция Display() для типа string
void Display(string t)
{
    // вывод типа string выделен отдельной функцией
    cout << "t = " << t.c_str() << endl;
}

int main()
{
    int a = 10;
    double d = 3.55;
    string s = "Hello";

    // вызов шаблонной функции
    Display(a); // t = 10
```

```
    Display(d); // t = 3.55

    // вызов "перегруженной" функции для типа string
    Display("abcd"); // t = abcd
    Display(s); // t = Hello

    return 0;
}
```

Шаблонный класс

Шаблон класса позволяет оперировать данными разных типов в общем. То есть, нет привязки к некоторому конкретному типу данных (int, float, …). Вся работа выполняется над некоторым обобщенным типом данных, например типом с именем T.

Фактически, объявление шаблона класса есть только описанием. Создание реального класса с заданным типом данных осуществляется компилятором в момент компиляции, когда объявляется объект класса.

Общая форма объявления шаблона класса и объекта шаблонного класса, не содержащего аргументов? Ключевое слово template

В простейшем случае общая форма объявления шаблона класса без аргументов имеет следующий вид:

```
template <class T> class ClassName
{
    // тело класса
    // ...
}
```

где

- T – обобщенное имя типа, который используется методами класса;
- ClassName – имя класса, который содержит методы оперирования обобщенным типом класса.

Общая форма объявления объекта шаблонного класса имеет следующий вид:

```
ClassName <type> objName;
```

где

- ClassName – имя шаблонного класса;
- type – конкретный тип данных в программе;
- objName – имя объекта (экземпляра) класса.

Ключевое слово class может быть заменено на слово typename. В этом случае общая форма объявления шаблонного класса может быть следующей:

```
template <typename T>
class ClassName
{
    // тело класса
    // ...
}
```

В данном случае нет разницы между использованием слов class и typename.

Преимущества использования шаблонов классовю.

Объявление шаблона класса дает следующие преимущества:

- избежание повторяемости написания программного кода для разных типов данных. Программный код (методы, функции) пишется для некоторого обобщенного типа T. Название обобщенного типа можно давать любое, например TTT;

- уменьшение текстовой части программного кода, и, как следствие, повышение читабельности программ;
- обеспечение удобного механизма передачи аргументов в шаблон класса с целью их обработки методами класса.

Объявление шаблона класса, содержащего методы обработки числа, тип которого может быть целым или вещественным

Пусть нужно объявить шаблон класса, который будет обрабатывать некоторое число. Число может быть любого типа, который позволяет выполнять над ним арифметические операции.

В примере объявляется шаблон класса, содержащий методы, которые выполняют следующие операции над некоторым числом:

- умножение числа на 2;
- деление одного числа на другое. Для целых типов выполняется деление нацело;
- возведение числа в квадрат (степень 2).

Объявление шаблона имеет вид

```
// шаблон класса, реализующего число разных типов
template <class T>
class MyNumber
{
   public:
   // конструктор
   MyNumber(void) { }

   // метод, умножающий число на 2
   void Mult2(T* t);

   // метод, возвращающий квадрат числа для некоторого типа T
   T MySquare(T);

   // метод, который делит два числа типа T и возвращает результат типа T
   T DivNumbers(T, T);
};

// реализация метода, умножающего число на 2
template <class T> void MyNumber<T>::Mult2(T* t)
{
   *t = (*t)*2;
}

// реализация метода, возвращающего квадрат числа
template <class T> T MyNumber<T>::MySquare(T number)
{
   return (T)(number*number);
}

// метод, который делит 2 числа и возвращает результат от деления
template <class T> T MyNumber<T>::DivNumbers(T t1, T t2)
{
   return (T)(t1/t2);
}
```

Использование шаблона класса MyNumber в другом программном коде

```
MyNumber <int> mi; // объект mi класса работает с типом int
MyNumber <float> mf; // объект mf работает с типом float

int d = 8;
float x = 9.3f;

// умножение числа на 2
mi.Mult2(&d); // d = 16
mf.Mult2(&x); // x = 18.6

// возведение числа в квадрат
int dd;
dd = mi.MySquare(9); // dd = 81 - целое число

double z;
z = mf.MySquare(1.1); // z = 1.21000... - вещественное число

// деление чисел
long int t;
float f;

t = mi.DivNumbers(5, 2); // t = 2 - деление целых чисел
f = mf.DivNumbers(5, 2); // f = 2.5 - деление вещественных чисел
```

Форма объявления шаблона класса, принимающего аргументы

Бывают случаи, когда в шаблоне класса нужно использовать некоторые аргументы. Эти аргументы могут использоваться методами, которые описываются в шаблоне класса.

Общая форма шаблона класса, содержащего аргументы, следующая:

```
template <class T, type1 var1, type2 var2, ..., typeN varN> class ClassName
{
   // тело шаблона класса
   // ...
}
```

где

- T – некоторый обобщенный тип данных;
- *type1, type2, …, typeN* – конкретные типы аргументов с именами *var1*, *var2*, …, *varN*;
- *var1, var2, …, varN* – имена аргументов, которые используются в шаблоне класса.

Общая форма объявления объекта шаблонного класса, содержащего один аргумент:

```
ClassName <type, arg> objName;
```

где

- ClassName – имя шаблонного класса;
- type – конкретный тип данных, для которого формируется реальный класс;
- arg – значение аргумента, которое используется в шаблоне класса;
- objName – имя объекта шаблонного класса.

Использования шаблона класса, принимающего два аргумента

В примере реализуется шаблон класса CMyArray, который содержит методы обработки массива чисел. Тип элементов массива может быть вещественным или целым.

Шаблон класса получает два целых числа:

- count – число элементов массива. Используется при инициализации класса с помощью конструктора с 1 параметром;
- num – число, служащее для проведения операций над массивом.

Эти числа используются в методах для выполнения операций над массивом. Шаблон класса содержит следующие данные и методы:

- количество элементов массива n;
- массив элементов (чисел) A заданной размерности (10);
- метод Power(), реализующий возведение элементов массива A в степень num. Значение num есть входным параметром (аргументом);
- метод CalcNum(), реализующий подсчет числа элементов, которые больше заданного параметра num.

Текст шаблона класса следующий:

```
// шаблон класса, получающего 2 параметра
template <class TT, int count, int num> class CMyArray
{
   private:
   int n; // число элементов массива
   TT A[10]; // массив элементов

   public:
   // конструктор класса без параметров
   CMyArray()
   {
      // число элементов берем из входного параметра count
      n = count;

      // заполнить массив произвольными значениями
      for (int i=0; i<n; i++)
         A[i] = (TT)(i*2);
   }

   // конструктор класса с 1 параметром
   CMyArray(int cnt)
   {
      if (cnt<=10) n = cnt;
      else n = 0;

      // заполнение массива произвольными значениями
      for (int i=0; i<n; i++)
         A[i] = (TT)(i*2);
   }

   // методы доступа
   int GetN(void) { return n; }

   void SetN(int n)
   {
      if (n<=10) this->n = n;
```

```
        else n=0;

        for (int i=0; i<n; i++)
            A[i] = (TT)(i*2);
    }

    // метод, возвращающий значение элемента массива с заданным индексом
    TT GetItem(int index) { return (TT)A[index]; }

    // методы, выполняющие операции над массивом A
    // возведение элементов массива в степень num
    void Power(void);

    // подсчет числа элементов, значения которых есть больше num
    int CalcNum(void);
};

// возведение значения элементов массива в степень num
template <class TT, int count, int num>
void CMyArray<TT, count, num>::Power(void)
{
    if (n<0) return;

    for (int i=0; i<n; i++)
        A[i] = System::Math::Pow(A[i], num);
}

// метод, определяющий число элементов массива,
// которые больше заданного числа num (num - входящий параметр)
template <class TT, int count, int num>
int CMyArray<TT, count, num>::CalcNum(void)
{
    int k = 0;

    // цикл подсчета
    for (int i=0; i<n; i++)
        if (A[i] > num)
            k++;
    return k;
}
```

Использование шаблона в некотором другом программном коде (функции, методе)

```
// использование шаблона класса CMyArray
// массив целых чисел, параметры: count=7, num=2
CMyArray <int, 7, 2> ai1;

// массив целых чисел, вызов конструктора с 1 параметром
CMyArray <int, 8, -3> ai2(6); // число элементов count = 6, num=-3

// массив вещественных чисел типа double, вызов конструктора без параметров
CMyArray <double, 4, 5> ad1;

// проверка
```

```
int n, t;
double x;

n = ai1.GetN(); // n = 7
n = ai2.GetN(); // n = 6
n = ad1.GetN(); // n = 4

// проверка массива
t = ai1.GetItem(3); // t = 6
t = ai2.GetItem(0); // t = 0
x = ad1.GetItem(2); // x = 4.0

// вызов методов обработки массива и проверка результата
ai1.Power(); // возведение элементов массива в степень пит=2
t = ai1.GetItem(3); // t = 6^2 = 36

// подсчет числа элементов, которые больше чем -3
// всего в массиве класса ai2 6 элементов
t = ai2.CalcNum(); // t = 6

// работа с классом, реализующим тип double
x = ad1.GetItem(3); // x = 6.0
ad1.Power(); // возведение чисел массива x в степень пит = 5
x = ad1.GetItem(3); // x = 6.0^5 = 7776
```